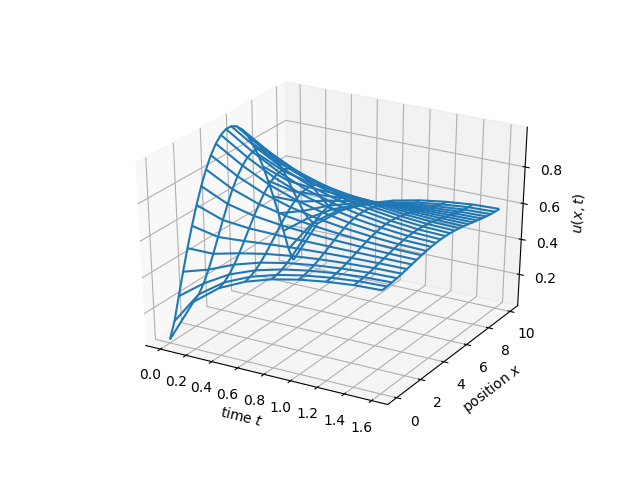
Andrew Durkiewicz Homework 11

Question 1

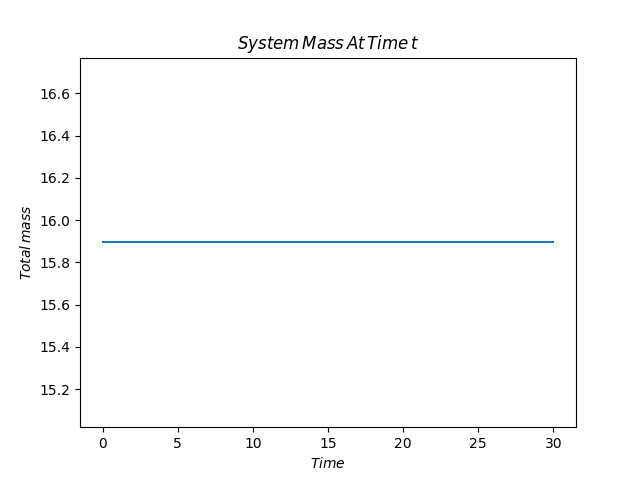
Part 1)

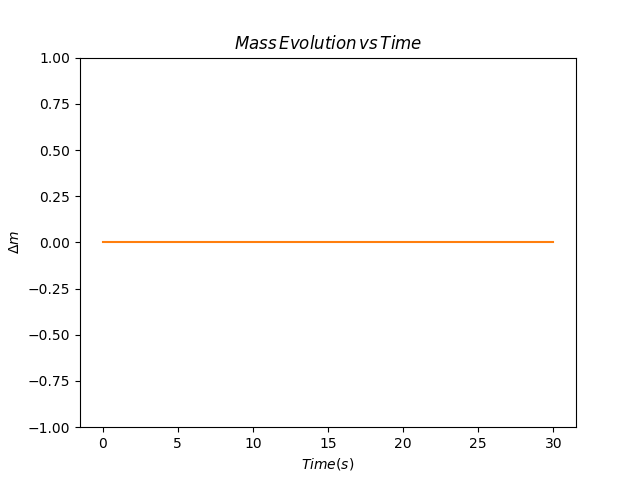
Below is a 3D graph of my solution for U(x,t):



Part 2)

The mass is the total sum of all the U(x\_n,t) values over the range of all n’s. Here is my plot of the total mass for each time t:





To show that the mass isn’t changing, I took the difference of the i’th t value for the mass M(x,t\_i) and took plotted its variance with the initial mass. So, as can be seen by the plot, there is no variance with the mass in time:

CODE for question 1:

#!/usr/bin/env python3

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

a = .5\*\*2

l = 10

T = 30

dx = .4

dt = .2

ld = a/dx\*\*2\*dt #lambda

Nx = int(1+round(10/dx))

Nt = 1 + round(T/dt)

x\_range = np.linspace(0,l,Nx)

t\_range = np.linspace(0,T,Nt)

def u0(x):

#initial conditions for when t = 0

return np.sin(np.pi \* x / l)

#initialize u matrix:

u = u0(x\_range)

#U\_{n+1} = A.u\_n

A = np.zeros((Nx,Nx))

#using the matrix form of our equation:

i = np.arange(Nx)

j = np.arange(Nx-1)

A[i,i] = 1-2\*ld

A[j,j+1] = ld

A[j+1,j] = ld

A[0,-1] = ld

A[-1,0] = ld

step = 20 # Plot every 10 time steps

plot\_data = np.zeros((Nt//step+2, Nx))

plt\_i = 0

masses = np.zeros(Nt)

for t\_i in range(Nt):

u = A.dot(u)

masses[t\_i] = np.sum(u) # Probably off by a scalar factor

if t\_i % step == 0 or t\_i == Nt-1:

plot\_data[plt\_i, :] = u

plt\_i += 1

graph = 0

while graph < 3:

if graph == 0:

# Setup plot

plt.plot(t\_range, masses)

plt.title(r'$System\/Mass\/At\/Time\/t$')

plt.xlabel(r'$Time$')

plt.ylabel(r'$Total\/mass$')

plt.savefig('mass\_at\_time\_t.png')

if graph == 2:

X, Y = np.meshgrid(range(plot\_data.shape[0]), range(plot\_data.shape[1]))

Z = plot\_data[X, Y]

fig = plt.figure()

ax = fig.add\_subplot(111, projection="3d")

ax.plot\_wireframe(X\*dt, Y\*dx, Z)

ax.set\_xlabel(r"time $t$")

ax.set\_ylabel(r"position $x$")

ax.set\_zlabel(r'$u(x,t)$')

plt.savefig("3d.png")

#lets find how much the mass deviates from the initial mass:

if graph == 1:

def mass\_change(m0,m):

return abs(m-m0)

change\_mass = np.array([])

for i in range(len(masses)):

change\_mass=np.append(change\_mass,(mass\_change(masses[0],masses[i])))

plt.plot(t\_range,change\_mass)

plt.ylim(-1,1)

plt.xlabel(r'$Time(s)$')

plt.ylabel(r'$\Delta m$')

plt.title(r'$Mass\/Evolution\/vs\/Time$')

plt.savefig('Mass\_Evolution\_vs\_time.png')

graph+=1

Question 2)

I solved the PDE using a 3D wireframe and used a contour:

|  |  |
| --- | --- |
| Δx = 0.2 , Δt = 0.1 | Δx = 0.1 , Δt = 0.2 |
|  |  |
|  |  |

The main difference that I notice is the way that peaks are occuring in U(x,t). For the δx = 0.2 and δt = 0.1, the peaks are much more smooth and less defined as the sharp peaks for δt = 0.2 and δx = 0.1.

Part C Below:

Part C)

|  |  |
| --- | --- |
| Δx = 1/10 | Δx = 1/30 |
|  |  |
| Δx = 1/70 | Δx = 1/200 |
|  |  |
| Δx = 1/550 | Δx = 1/1500 |
|  |  |

The accuracy order for each δx printed as such:

Order Error:

For dx = 1/10, the resulting accuracy is: 0.879133761846

For dx = 1/30, the resulting accuracy is: 1.11005002514

For dx = 1/70, the resulting accuracy is: 1.00479510211

For dx = 1/200, the resulting accuracy is: 1.00946343235

For dx = 1/550, the resulting accuracy is: 1.05517391856

CODE for problem2 parts a-b:

#!/usr/bin/env python3

import numpy as np

from matplotlib import pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.cm as cm

import matplotlib.mlab as mlab

error = []

## Initial conditions

T = 1

x0, x1 = 0, 1

dt = .2

dx = .1

c = 1

def u\_t0(x):

# Initial positions

return np.sin(2 \* np.pi \* x)

def v\_t0(x):

# Initial velocities

return 2 \* np.pi \* np.sin(2 \* np.pi \* x)

## Setup Parameters

intX = int(1 + round((x1-x0) / dx))

intT = 1 + round(T / dt)

x\_range = np.linspace(x0, x1, intX)

t\_range = np.linspace(0, T, intT)

#use our t values to make our u matrix

u\_prev = u\_t0(x\_range) # t\_i=0

u\_mat = u\_prev + dt \* v\_t0(x\_range) # t\_i=1

# u\_{n+1} = A (dot) u\_{n}

A = np.zeros((intX, intX))

range1 = np.arange(intX)

range2 = np.arange(intX-1)

ld = c\*\*2 \* dt\*\*2 / dx\*\*2

A[range1, range1] = 2 - 2\*ld

A[range2, range2+1] = ld

A[range2+1, range2] = ld

A[0, -1] = ld # Coefficients for periodic boundary condition

A[-1, 0] = ld

plot\_data = np.zeros((intT, intX))

plot\_data[0, :] = u\_prev

def solution(x, t):

return np.sin(2\*np.pi\*x) \* (np.cos(2\*np.pi\*t) + np.sin(2\*np.pi\*t))

plot\_our\_solution = np.zeros\_like(plot\_data)

## Solve - iterate over time

for t\_i in range(intT-1):

u\_prev[:], u\_mat = u\_mat, A.dot(u\_mat) - u\_prev

plot\_data[t\_i, :] = u\_mat

plot\_our\_solution[t\_i, :] = solution(x\_range, t\_range[t\_i])

error.append(np.max(np.abs(plot\_data - plot\_our\_solution)))

# Setup plot

X, Y = np.meshgrid(range(plot\_our\_solution.shape[0]), range(plot\_our\_solution.shape[1]))

Z = plot\_our\_solution[X, Y]

fig = plt.figure()

ax = fig.add\_subplot(111, projection="3d")

ax.plot\_wireframe(X\*dt, Y\*dx, Z)

ax.set\_xlabel(r"time $t$")

ax.set\_ylabel(r"position $x$")

ax.set\_title(r'$\Delta x\/=\/10^{-1}\/,\/t\/=\/2 \/ \cdot \/10^{-1}$')

plt.savefig('1b\_final\_true.png')

plt.show()

CODE for part c:

#!/usr/bin/env python3

import numpy as np

from matplotlib import pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

dx\_range = (1/10, 1/30, 1/70, 1/200, 1/550, 1/1500)

dx\_range\_string = ['1/10','1/30','1/70','1/200','1/550','1/1500']

error = []

for dx in dx\_range:

## Initial conditions

T = 1

x0, x1 = 0, 1

dt = dx \* .9

c = 1

def u\_t0(x):

# Initial positions

return np.sin(2 \* np.pi \* x)

def v\_t0(x):

# Initial velocities

return 2 \* np.pi \* np.sin(2 \* np.pi \* x)

## Setup Parameters

intX = int(1 + round((x1-x0) / dx))

intT = 1 + round(T / dt)

x\_range = np.linspace(x0, x1, intX)

t\_range = np.linspace(0, T, intT)

#use our t values to make our u matrix

u\_prev = u\_t0(x\_range) # t\_i=0

u\_mat = u\_prev + dt \* v\_t0(x\_range) # t\_i=1

# u\_{n+1} = A (dot) u\_{n}

A = np.zeros((intX, intX))

range1 = np.arange(intX)

range2 = np.arange(intX-1)

ld = c\*\*2 \* dt\*\*2 / dx\*\*2

A[range1, range1] = 2 - 2\*ld

A[range2, range2+1] = ld

A[range2+1, range2] = ld

A[0, -1] = ld # Coefficients for periodic boundary condition

A[-1, 0] = ld

plot\_data = np.zeros((intT, intX))

plot\_data[0, :] = u\_prev

def solution(x, t):

return np.sin(2\*np.pi\*x) \* (np.cos(2\*np.pi\*t) + np.sin(2\*np.pi\*t))

plot\_our\_solution = np.zeros\_like(plot\_data)

## Solve - iterate over time

for t\_i in range(intT-1):

u\_prev[:], u\_mat = u\_mat, A.dot(u\_mat) - u\_prev

plot\_data[t\_i, :] = u\_mat

plot\_our\_solution[t\_i, :] = solution(x\_range, t\_range[t\_i])

error.append(np.max(np.abs(plot\_data - plot\_our\_solution)))

# Setup plot

if dx == 1/70:

X, Y = np.meshgrid(range(plot\_our\_solution.shape[0]), range(plot\_our\_solution.shape[1]))

Z = plot\_our\_solution[X, Y]

fig = plt.figure()

ax = fig.add\_subplot(111, projection="3d")

ax.plot\_wireframe(X\*dt, Y\*dx, Z)

ax.set\_xlabel(r"time $t$")

ax.set\_ylabel(r"position $x$")

plt.savefig('Ex-solution.png')

print('Order Error:')

for i in range(len(error)-1):

print("For dx = %s, the resulting accuracy is: "%dx\_range\_string[i],error[i+1]/error[i] / (dx\_range[i+1]/dx\_range[i]))